**Explain Django Architecture or how it works ?**

Django follows the MVT /MTV(Model View Template) pattern which is based on the Model View Controller architecture.

Template --- presentation layer, html pages, images , DTL etc

Model – linked with db classes

View --- business logic to display/capture/update/delete data

**Explain the django project directory structure?**

* manage.py - A command-line utility that allows you to interact with your Django project
* \_\_init\_\_.py - An empty file that tells Python that the current directory should be considered as a Python package
* settings.py - Comprises the configurations of the current project like DB connections.
* urls.py - All the URLs of the project are present here
* wsgi.py - This is an entry point for your application which is used by the web servers to serve the project you have created.

**What are models in Django?**

A model in Django refers to a class that maps to a database table or database collection. Each attribute of the Django model class represents a database field.

**What is ORM?**

ORM is an acronym for the object-relational mapper.

The ORM’s main goal is to transmit data between a relational database and application model.

It avoids writing raw queries, it is possible to retrieve, save, delete and perform other operations over the database without even writing any SQL query

ORM, as from the name, maps objects attributes to respective table fields.

**Advantages of ORM**

1. Rapid development
2. Makes project more portable
3. We can easily change to DB.

**Define static files and explain their uses?**

In Django static files refer to additional files such as images. Javascript or CSS. Apart from that Django provides django.contrib.staticfiles to manage these static files.

**What is django-admin and manage.py and explain its commands?**

django-admin is Django’s command-line utility for administrative tasks. In addition to this, a manage.py file is also automatically created in each Django project.

Some common commands are –-

* django-admin dumpdata
* django-admin migrate
* django-admin makemigrations
* django-admin migrate
* django-admin shell
* django-admin startproject
* django-admin createsuperuser
* python manage.py dumpdata <app\_name>.<model\_name> --format format\_value --indent int –output filename
* django-admin changepassword

**What are Django URLs?**

URLs are one of the most important parts of a web application and Django provides you with an elegant way to design your own custom URLs with help of its module known as URLconf (URL Configuration). The basic functionality of this python module is to map our view (which contains business logic) with incoming request/response.

**What is the difference between a project and an app in Django**

In simple words Project is the entire Django application and an app is a module inside the project that deals with one specific use case

**What are different model inheritance styles in the Django?**

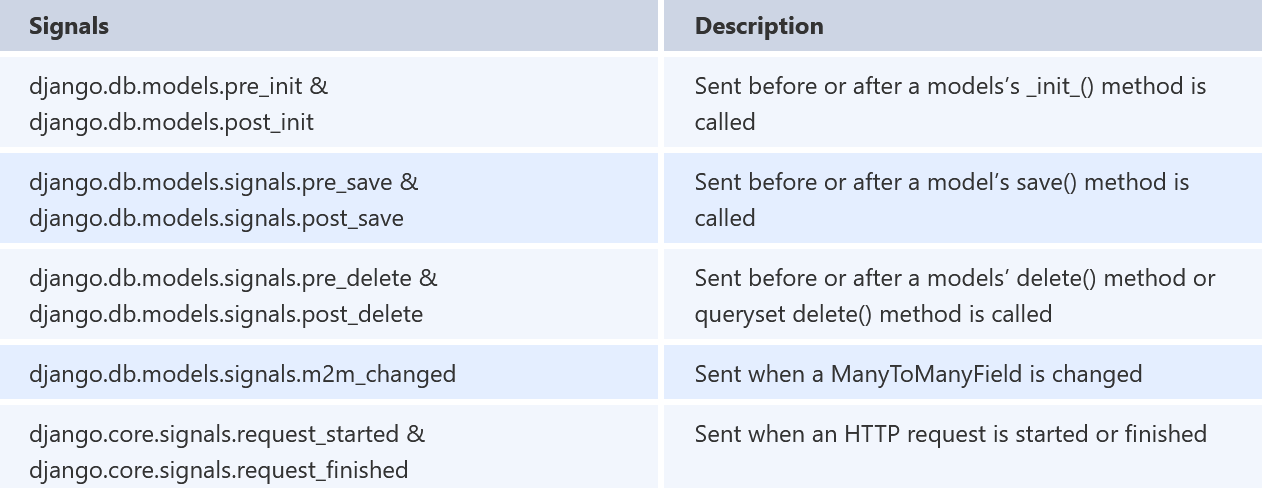
There are mainly three types of model inheritance in Django –

* Abstract Base Class Inheritance
* Multi-table inheritance
* Proxy Model inheritance

**What are Django Signals**

The signals are the utilities that allow us to associate events with actions. It can built-in or custom.

Some built-in signals are ---



**Explain the caching strategies in the Django?**

Caching refers to the technique of storing the output results when they are processed initially so that next time when the same results are fetched again, instead of processing again those already stored results can be used, which leads to faster accessing as well us less resource utilization.

Caching can be broadly classified in two parts—

1. Where to store cache info—
2. Database ---- cache data stored in database
3. Local memory---- stored in local memory **(faster)**
4. File System caching --- cached data is stored in file

**Note:**

If using database caching then we should run below command so that cache table is created

*python manage.py createcachetable*

1. What to cache
2. Site caching --- caching whole project
3. Per view caching --- caching each view function/class
4. Template caching

**How to configure static files?**

Ensure that django.contrib.staticfiles is added to your INSTALLED\_APPS

In your settings file. define STATIC\_URL for ex.

STATIC\_URL = '/static/'

Define the 'STATICFILES\_DIRS' (path where you want to store static files) in case if not keeping in applications.

While using them in html page load them on top. {% load static %}

**Explain Django Response lifecycle?**

On very overview level we can define it as---

Request ----> Goes to view class/function, routed/mapped by urls.py file , after doing some operation, returns HttpResponse ----> template

Whenever a request is made to a web page, Django creates an HttpRequest object that contains metadata about the request. After that Django loads the particular view, passing the HttpRequest as the first argument to the view function.

Each view will be returning an HttpResponse object.

On the big picture following steps occur when a request is received by Django:

* First of the Django settings.py file is loaded which also contain various middleware classes (MIDDLEWARES)
* The middlewares are also executed in the order in which they are mentioned in the MIDDLEWAREST
* From here on the request is now moved to the URL Router, who simply gets the URL path from the request and tries to map with our given URL paths in the urls.py.
* As soon as it has mapped, it will call the equivalent view function, from where an equivalent response is generated
* The response also passes through the response middlewares and send back to the client/browser.

Request---->middleware -----> urls ----> view function/class ----> httpResponse

**What databases are supported by Django?**

PostgreSQL and MySQL, SQLite and Oracle.

Apart from these, Django also supports databases such as ODBC, Microsoft SQL Server, IBM DB2, SAP SQL Anywhere, and Firebird using third-party packages. Note: Officially Django doesn’t support any no-SQL databases.

**What's the use of a session framework?**

A session is a mechanism to store information on the server side during the interaction with the web application.

The session is a semi-permanent (depending on age of session) and two-way communication between the server and the browser

If using session then django.contrib.sessions must be installed in installed\_app section.

We can store session data in three ways—

* Database backed session
* File based backed session
* Local memory session/ cache backed session
* Cookies backed session

**Note:**

If using database-backed session then we need to run migrate command so that session table is created.

**What is context in the Django?**

Context is a dictionary mapping template variable name given to Python objects in Django. This is the general name, but you can give any other name of your choice if you want.

Context is used to pass data/value from view to template.

**What is django.shortcuts.render function?**

Combines a given template with a given context dictionary and returns an HttpResponse object with that rendered text.

The render() shortcut renders templates with a request context. Template context processors take the request object and return a dictionary which is added to the context

If don’t want to render with request then use – request=None.

**What’s the significance of the settings.py file?**

As the name suggests this file stores the configurations or settings of our Django project, like database configuration, backend engines, middlewares, installed applications, main URL configurations, static file addresses, templating engines, main URL configurations, security keys, allowed hosts, and much more.

**How to view all items in the Model?**

Modelname.objects.all()

**How to filter items in the Model?**

Modelname.objects.filter(fieldname=value)

We can use field-lookup with fields for more refined result.

**How to use file-based sessions**

To use file- based session add below two lines in settings.py file.

SESSION\_ENGINE='django.contrib.sessions.backends.file'

SESSION\_FILE\_PATH=full\_path\_for\_session\_file

SESSION\_EXPIRE\_AT\_BROWSER\_CLOSE= 'bool' ---- to make session expired on browser closed.

**What is mixin?**

Its type of multiple inheritances where in you can combine behaviors and attributes of more than one parent class. It provides us with an excellent way to reuse code from multiple classes.

**Drawback:**

It becomes difficult to analyze what a class is doing and which methods to override in case of its code being too scattered between multiple classes.

|  |  |
| --- | --- |
| **Mixin** | **Multiple inheritance** |
| >Parent class instantiation not possible or of no use  >Parent class contain only instance method not instance variable.  >Methods are useful only for Child class.  >Parent class should be direct child class of object class. | >We can instantiate.  >It may contain both instance variable and method and static methods and variable.  >Methods are useful for parent and child class.  >Parent class can inherit any other class also |

Example:

Let say we are working with djano views where we need to return Json response without using REST API method, we can use mixin in below way –

Defining own mixin file--

import json

from django.http import JsonResponse

class JsonMininx:

    def json\_data(self,data):

        return JsonResponse(data)

views.py file ( use that mixin where ever required)

class JsonCBV(JsonMininx,View):

    def get(self,request):

        d={1:"hello",2:"Shyam",0:"Hare Hare"}

        return self.json\_data(d)

**Difference between Django OneToOneField and ForeignKey Field?**

The only difference between these two is that ForeignKey field consists of on\_delete option along with a model’s class because it’s used for many-to-one relationships

On the other hand, the OneToOneField, only carries out a one-to-one relationship and requires only the model’s class.

**How to get a particular item in the Model**

Modelname.objects.filter(fieldname=value)

Modelname.objects.get(id=value)

**How to obtain the SQL query from the queryset?**

Queryset.query

**Difference between select\_related and prefetch\_related?**

**Explain Q objects in Django ORM?**

Q objects are used to write complex queries, as in filter() functions sometime we need to use AND, OR, NOT condition that can be easily achieved as ---

* filter(Q(condition1)&Q(condition2))
* filter(Q(condition1)|Q(condition2))
* filter(~Q(condition))

**What are Django exceptions**

django.core.exceptions package --- it has exception like middleware or ORM exception, few of them are---

EmptyResultset

MulipltObjectFoundExcception

ObjectDoesNotExists

URL resolver Exception – it has below exception classes

Resolver404

NoReverseMatch

**Purpose of apps.py file:**

This file is created to help the user include any application configuration for the app

**Primary key and Unique key**

**Primary key**

The primary key is a **unique or non-null key** that uniquely identifies every record in that table or relation. Primary key can never be NULL or duplicated.

* A table can’t contain more than one column having primary key attribute.
* A primary key column of one table can be referenced by a foreign key column of another table.

**Unique key**

The unique key is a single column or combination of columns in a table to uniquely identify database records.

A unique key prevents from storing duplicate values in the column.

* A table can contain multiple unique key columns, unlike a primary key column.
* A unique key column can contain one NULL value for a column.

**What is Signal and its type**

Signals are utilities that can be used to bind any even with some action. These are majorly two type:

1. Built-in signa
2. Custom signal

**How to create signs/Custom signal**

We should create signal in a separate file.

my\_sig=Signal(providing\_args=data\_in\_list\_form) –------- here signal my\_sig is created

**How to send custom signal and handle them**

For sending custom signal we can use send() on singal name

signal\_name.send(sender,\*\*kwargs) -------------------- # Sending signal

@receiver(signalname) # Receiving signal

def my\_function():

#

**How to handle built-in signal**

We need to use @receiver decorator to handle signal on top of receiver function

@receiver(signal\_name, sender, \*\*kwargs)

def my\_function(sender,user,instance,\*\*kwargs)

#

**What are types of model inheritance**

1. Abstract base model inheritance ------ abstract=True
2. Multiple Table inheritance --------------- it’s like one parent , one child inheritance
3. Proxy model inheritance ----------------- proxy=True
4. Multilevel inheritance ------------------- it’s grandchild, father, grandfather

**What is middleware in Django**

It’s plugin/utilities which is processed during request/response cycle.

Its major role is session management, security, authentication etc.

If using any middleware (function or class) then that need to be added in settings.py file.

There are majorly two types of middleware

1. Class based middleware
2. Function based middleware

**Function based middleware**

def my\_middleware(**get\_response**):

#One-time config and initilization code

def my\_function(**request**):

#code to execute for each request/response before view called

**response=get\_response(request)**

#code to be executed for each request/response after view is called

return response

return my\_function

**Function based decorator in python**

def my\_decorator(func):

#some codes

def wrapper(\*args,\*\*kwargs):

#some codes or actions

return func(\*args,\*\*kw

**Class based middleware**

class MyMiddleware:

**def \_\_init\_\_(self,get\_response):**

self.get\_response=get\_response

#One-time configuration and initialization

**def \_\_call\_\_(self,request):**

#Code to be executed for each request before the view are called

**response=self.get\_response(request)**

#code to be executed for each request/response after the view is called

return response

**class Based Decorator:**

class MyDecorator:

def \_\_init\_\_(self,func):

self.func=func

def \_\_call\_\_(self, \*args,\*\*kwds):

return self.func(\*args,\*\*kwargs)